Synopsys Users Group
AUSTIN 2011

# Object Oriented State Machines in System Verilog

VJ Sananda
AMD

## Outline

Synopsys Users Group
AUSTIN 2011

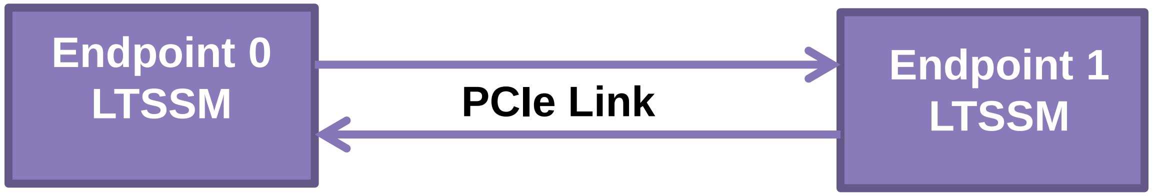
- Introduction
- PCIe Link Training & Status State Machine (LTSSM)
- Principle of Operation: Object Oriented State Machine
- Object Oriented LTSSM Transactor
- Results
- Conclusions

## Introduction

- Bus Functional Models / Transactors interacting with Design have State Machines
- Traditionally described using State Variable + Control Flow code for transitions
- This talk : An Object Oriented way to model a State Machine for a Transactor
  - Not for synthesizable code !
  - Case study using PCIe Link Training State Machine
  - Demonstrate how state machine behavior can be changed in order to test design

## PCIe LTSSM

- PCIe devices communicate via serial point to point connections (lanes) with embedded clocks
- Link Training → Auto-negotiation protocol to:
  - Establish symbol lock (receiver recovers clock)
  - Configure physical lanes (aggregate) to form links
  - Agree on speed and protocol version
- PCIe Link Training and Status State Machines
  - One in each endpoint
  - Exchange symbol sequences → State transitions until Link is up.



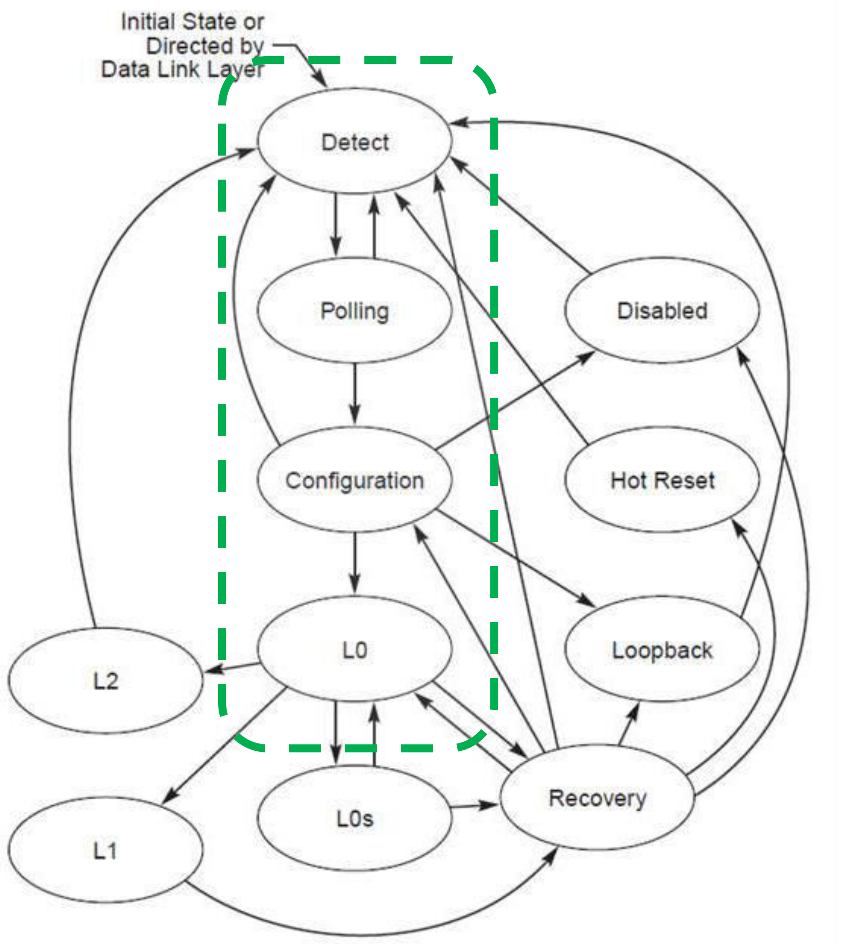
## PCIe LTSSM

- PCIe packet based
- During Link Training
  - Tx & Rx Symbol sequences (called Ordered sets)
  - 2 Types: **TS1** & **TS2**
  - **IDLE** sequence after Link is up
- For these sequences Transactor uses class hierarchy →

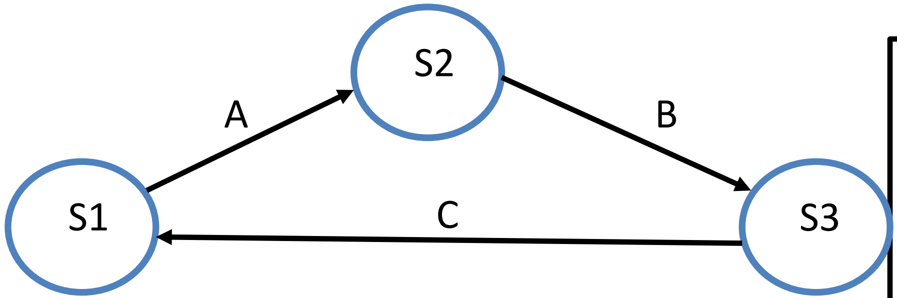
```
typedef enum {TS1,TS2,IDLE,OTHER}
      OSType ;

virtual class OrderedSet ;
  virtual function OSType getType() ;
  endfunction
endclass

class OrderedSet_TS1 extends
      OrderedSet ;
 function OSType getType();
   return (TS1) ;
 endfunction
endclass
………. Similar definition for TS2, IDLE
```

# PCIe LTSSM : State Diagram

- For this case study focus on link bring up,
  - reaching L0 after reset
- Each state characterized by
  - Tx Ordered sets (specific Type)
  - Rx Ordered sets (certain type and number) → Transition to next state
  - Timeout if state transition criteria not met
  - Sub-states ( example: Polling.Active, Polling.Config )



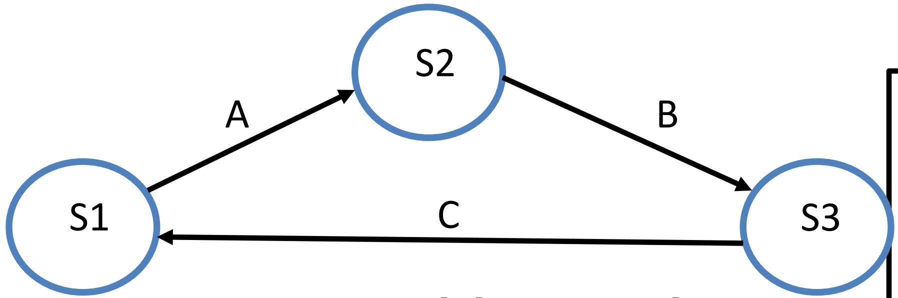
# Object Oriented State Machine: Principles of Operation



- State transitions when events A,B & C received
- "Traditional" implementation: Uses control logic (case , if-then-else)

```
Case (CurrentState)
 S1:  { Print "I am in State S1";
          if (input_event  == A)
             NextState = S2 ; }

 S2:  { Print "I am in State S2";
          if (input_event  == B)
             NextState = S3; }

 S3:  { Print "I am in State S3";
          if (input_event  == C)
             NextState = S1 ; }
EndCase

@ (state_transition_boundary)
     CurrentState = NextState ;
```

# Object Oriented State Machine: Principles of Operation



- State transitions when events A,B & C received
- State definitions: inherit from Base_State with virtual methods to:
  - Process input
  - Show output

```
class Base_State {
   Base_State  NextState ;
   Virtual Method WaitForInput(event _e)
      ReturnType Base_State ;
   Virtual Method Output();
}
class S1_State inherits Base_State {
   Method WaitForInput (event _e) {
      If (_e == A)
         return NextState ;
   }
   Method Output() {
     Print "I am in State S1";
   }
}
```

Pseudo-code

# Object Oriented State Machine: Principles of Operation

- NextState member variables hold state transition information
- CurrState is a Base_State handle initialized to S1
- Rely on polymorphism to call the appropriate WaitForInput() and Output() method for each state

```
class FSM {
 //Declarations .........
 Constructor() {
   //Define Next state transitions
   S1.NextState  = S2 ; ......
   //Initial State
   currState = S1;
  }
 //Main State Machine Loop
 Method run() {
  Forever {
        currState.Output();
         nxtState = currState.WaitForInput();
         currState = nxtState ;
} } }
```

# Object Oriented LTSSM Transactor

Synopsys Users Group
AUSTIN 2011

```
virtual class Base_State ;
  int TransmitCnt ;
  int Timeout ;
  int TransitionCnt ;
  Base_State NextState ;
  Base_State TimeoutState ;
  virtual task Transmit ();
  endtask
  virtual task Receive (ref Base_State
      _state);
  endtask
  virtual task ReceiveTimeout (
              ref Base_State _state);
      repeat(Timeout) @(posedge clk);
        _state = TimeoutState;
  endtask
  task display() ;
      $display($stime,"::Endpoint%d::%s",
      PhyEndpoint,StateDescr);
  endtask
endclass
```

- Like simple example, all LTSSM States derive from a **Base_State**
- **TransmitCnt**: Number of Ordered sets Tx
- **TransitionCnt**: Number of Ordered sets Rx for a state transition to occur
- **NextState** and **TimeoutState** hold the states to Transition to.
- virtual task implementations for each state called within Main FSM loop.

# Object Oriented LTSSM Transactor

Synopsys Users Group
AUSTIN 2011

```
//State Definition Example
class Configuration_Idle_State extends
      Base_State ;
 function new () ;
   TransitionCnt = 8;
   TransmitCnt   = 16 ;
   StateDescr = "Configuration Idle" ;
 endfunction
task Transmit();
  Idle idle = new ;
  repeat (TransmitCnt)
      begin
         send(idle);
         @(posedge clk);
     end
endtask
```

Set member variables in constructor

```
task Receive(ref Base_State _state);
    while(1)
      begin
        @(posedge clk);
        if  (rx_ts.getType() == IDLE )
              count++;
        else
               count=0;
         if (count == TransitionCnt) break ;
      end
    //Transition to next state
    _state = NextState;
  endtask

endclass
```

Implement virtual tasks

# Object Oriented LTSSM Transactor

Synopsys Users Group
AUSTIN 2011

```
//Class definition of LTSSM
class LTSSM ;
 function new (Phy _phy, bit
    endpoint ) ;
  Detect = new (_phy,endpoint);
  fsmState = Detect ; //Initial State

  Detect.NextState = PollingActive ;
  Detect.TimeoutState = Detect   ;
  ConfigurationIdle.NextState = L0 ;
  ConfigurationIdle.TimeoutState =
    Detect   ;
  . . . .
endfunction
```

Set Next state values (Regular Rx & Timeout)

```
task run () ;
 while( Enable )
  begin
    fork
      fsmState.Transmit() ;
      fsmState.display();
      begin
        fork
          fsmState.Receive(nxtState);
          fsmState.ReceiveTimeout(nxtState);
        join_any
       disable fork;
      end
    join
    fsmState = nxtState ;
  end
endtask
```

Tx and Rx threads run concurrently

Code block terminates when valid sequence Rx or Timeout

# Results : Test Rig

Synopsys Users Group
AUSTIN 2011

```
//Test Rig to exercise LTSSM
program test( input clk );
 initial
   begin
   //Instantiate Phy
    Phy phy_inst = new ;

   //Instantiate LTSSMs,define endpoints
     LTSSM design = new(phy_inst,0);
     LTSSM model  = new(phy_inst,1);

     fork
        design.run();
        model.run();
     join_none

 fork
   //Wait for link to come up
    while (~model.isLinkUp() )
          @(posedge clk);

    //Time out counter for Link bringup
      begin
        repeat (50000) @(posedge clk);
        $display($stime,"::ERROR:Link
                    Failed to come up");
      end
 join_any
 disable fork;

 $finish;
 end
endprogram
```

1 LTSSM mimics RTL & not modified

Test Terminates when link comes up or times out after 50000 clks

# Results : Output

Synopsys Users Group
AUSTIN 2011

```
Chronologic VCS simulator copyright 1991-2008
Contains Synopsys proprietary information.
Compiler version B-2008.12-B-E9; Runtime version B-2008.12-B-E9;  Jun 23 19:31 2011
        0::Endpoint0::Detect
        0::Endpoint1::Detect
        5::Endpoint1::Polling Active
        5::Endpoint0::Polling Active
    10245::Endpoint0::Polling Configuration
    10245::Endpoint1::Polling Configuration
    16685::Endpoint0::Configuration LinkLane
    16685::Endpoint1::Configuration LinkLane
    17385::Endpoint1::Configuration Complete
    17385::Endpoint0::Configuration Complete
    17545::Endpoint0::Configuration Idle
    17545::Endpoint1::Configuration Idle
    17705::Endpoint1::L0 , Link up
$finish called from file "test1.sv", line 43.
$finish at simulation time             17705
```

Link bring up with matched LTSSM transactors

With identical transactors at both ends, state transitions occur in lock step

# Results: Change transactor param

Synopsys Users Group
AUSTIN 2011

```
program test( input clk );
. . . .
      LTSSM model  = new(phy_inst,1);
      model.PollingActive.TransmitCnt = 0 ; //Change from 1024
      fork
        model.run();
. . . .
endprogram
```

Changing the number of ordered sets transmitted.

```
    5::Endpoint1::Polling Active
    5::Endpoint0::Polling Active
   85::Endpoint1::Polling Configuration
10245::Endpoint0::Polling Configuration
. . . .
12565::Endpoint0::L0
12625::Endpoint1::L0 , Link up
```

Link still comes up.
Enough Ordered Sets Tx by model in Polling Configuration.

Demonstrates protocol robustness

# Results: Change transactor param

```
model.PollingActive.TransmitCnt = 0 ; //Change from 1024
model.PollingConfiguration.TransmitCnt = 0 ; //Change from 16
```

```
   0::Endpoint0::Detect
   0::Endpoint1::Detect
   5::Endpoint1::Polling Active
   5::Endpoint0::Polling Active
  85::Endpoint1::Polling Configuration
40005::Endpoint0::Polling Active Timeout
40005::Endpoint0::Detect
40015::Endpoint0::Polling Active
. . . . .
481085::Endpoint1::Polling Active
481165::Endpoint1::Polling Configuration
499995::ERROR:Link Failed to come up
```

Shut off Ordered Set Transmission in 2 states

Repeated time outs

Link up failure

## Results: Change state transition

```
model.PollingConfiguration.NextState = model.ConfigurationIdle;
```

```
    0::Endpoint0::Detect
    0::Endpoint1::Detect
    5::Endpoint1::Polling Active
10245::Endpoint1::Polling Configuration
16685::Endpoint0::Configuration LinkLane   Test changes transition
16685::Endpoint1::Configuration Idle
49995::Link up Failed => wrong Nextstate transition,Fixing..
56685::Endpoint1::Configuration Idle Timeout
56685::Endpoint0::Configuration LinkLane Timeout
56685::Endpoint1::Detect
56685::Endpoint0::Detect
56695::Endpoint0::Polling Active
74075::Endpoint1::Configuration Complete
74235::Endpoint1::Configuration Idle
74235::Endpoint0::Configuration Idle
74395::Endpoint1::L0 , Link up
```

**Recovery after spurious next state transition**

# Results: Randomization example

Synopsys Users Group
AUSTIN 2011

**Constrained random testing by randomizing member variables and next state transition assignments.**

```
program test( input clk );
. . . .
  //Randomize TransmitCnt in Polling Active
  std::randomize (TransmitCnt) with {TransmitCnt inside { 0,16 }; } ;
  model.PollingActive.TransmitCnt = TransmitCnt ;

  randcase //Randomize State Transition
    //Correct transition 90% of the time
    90: model.PollingConfiguration.NextState = model.ConfigurationLinkLane;

    //Incorrect transition 10% of the time
   10: model.PollingConfiguration.NextState = model.ConfigurationIdle;
  endcase
endprogram
```

## Conclusions

- The PCIe LTSSM is a complex state machine
  - State transitions driven by Rx of ordered sets and timeout conditions
- Object Oriented transactor modeling the LTSSM:
  - Encapsulates code capturing behavior in each state, easier to maintain and extend.
  - Can vary behavior using member variable assignments.
  - Can setup arbitrary state transitions (to test error recovery and protocol robustness).
  - Natural fit for a constrained random test environment.
  - Complete source code example in paper.